# JPA

## JPA Component

The **jpa** component enables you to store and retrieve Java objects from persistent storage using EJB 3's Java Persistence Architecture (JPA), which is a standard interface layer that wraps Object/Relational Mapping (ORM) products such as OpenJPA, Hibernate, TopLink, and so on.

Maven users will need to add the following dependency to their `pom.xml` for this component:

xml<dependency> <groupId>org.apache.camel</groupId> <artifactId>camel-jpa</artifactId> <version>x.x.x</version> <!-- use the same version as your Camel core version --> </dependency>

Sending to the endpoint

You can store a Java entity bean in a database by sending it to a JPA producer endpoint. The body of the *In* message is assumed to be an entity bean (that is, a POJO with an @Entity annotation on it) or a collection or array of entity beans.

If the body is a List of entities, make sure to use **entityType=java.util.ArrayList** as a configuration passed to the producer endpoint.

If the body does not contain one of the previous listed types, put a Message Translator in front of the endpoint to perform the necessary conversion first.

From **Camel 2.19** onwards you can use **query**, **namedQuery** and **nativeQuery** option for the producer as well to retrieve a set of entities or execute bulk update/delete.

### Consuming from the endpoint

Consuming messages from a JPA consumer endpoint removes (or updates) entity beans in the database. This allows you to use a database table as a logical queue: consumers take messages from the queue and then delete/update them to logically remove them from the queue.

If you do not wish to delete the entity bean when it has been processed (and when routing is done), you can specify `consumeDelete=false` on the URI. This will result in the entity being processed each poll.

If you would rather perform some update on the entity to mark it as processed (such as to exclude it from a future query) then you can annotate a method with @Consumed which will be invoked on your entity bean when the entity bean when it has been processed (and when routing is done).

From **Camel 2.13** onwards you can use @PreConsumed which will be invoked on your entity bean before it has been processed (before routing).

If you are consuming a lot (100K+) of rows and experience OutOfMemory problems you should set the maximumResults to sensible value.

**Note:** Since **Camel 2.18**, JPA now includes a `JpaPollingConsumer` implementation that better supports Content Enricher using `pollEnrich()` to do an on-demand poll that returns either none, one or a list of entities as the result.

### URI format

jpa:entityClassName[?options]

For sending to the endpoint, the *entityClassName* is optional. If specified, it helps the Type Converter to ensure the body is of the correct type.

For consuming, the *entityClassName* is mandatory.

You can append query options to the URI in the following format, `?option=value&option=value&...`

### Options

| Name | Default Value | Description |
|---|---|---|
| `entityType` | *entityClassName* | Overrides the *entityClassName* from the URI. |
| `persistenceUnit` | `camel` | The JPA persistence unit used by default. |
| `consumeDelete` | `true` | **JPA consumer only:** If `true`, the entity is deleted after it is consumed; if `false`, the entity is not deleted. |
| `consumeLockEntity` | `true` | **JPA consumer only:** Specifies whether or not to set an exclusive lock on each entity bean while processing the results from polling. |
| `flushOnSend` | `true` | **JPA producer only:** Flushes the EntityManager after the entity bean has been persisted. |

| | | |
|---|---|---|
| `maximumResults` | `-1` | **JPA consumer only:** Set the maximum number of results to retrieve on the Query. **Camel 2.19:** it's also used for the producer when it executes a query. |
| `transactionManager` | `null` | This option is Registry based which requires the # notation so that the given `transactionManager` being specified can be looked up properly, e.g. `transactionManager=#myTransactionManager`. It specifies the transaction manager to use. If none provided, Camel will use a `JpaTransactionManager` by default. Can be used to set a JTA transaction manager (for integration with an EJB container). |
| `consumer.delay` | `500` | **JPA consumer only:** Delay in milliseconds between each poll. |
| `consumer.initialDelay` | `1000` | **JPA consumer only:** Milliseconds before polling starts. |
| `consumer.useFixedDelay` | `false` | **JPA consumer only:** Set to `true` to use fixed delay between polls, otherwise fixed rate is used. See ScheduledExecutorService in JDK for details. |
| `maxMessagesPerPoll` | `0` | **JPA consumer only:** An integer value to define the maximum number of messages to gather per poll. By default, no maximum is set. Can be used to avoid polling many thousands of messages when starting up the server. Set a value of 0 or negative to disable. |
| `consumer.query` | | **JPA consumer only:** To use a custom query when consuming data. |
| `consumer.namedQuery` | | **JPA consumer only:** To use a named query when consuming data. |
| `consumer.nativeQuery` | | **JPA consumer only:** To use a custom native query when consuming data. You may want to use the option `consumer.resultClass` also when using native queries. |
| `consumer.parameters` | | **Camel 2.12: JPA consumer only:** This option is Registry based which requires the # notation. This key/value mapping is used for building the query parameters. It's is expected to be of the generic type `java.util.Map<String, Object>` where the keys are the named parameters of a given JPA query and the values are their corresponding effective values you want to select for. |
| `consumer.resultClass` | | **Camel 2.7: JPA consumer only:** Defines the type of the returned payload (we will call `entityManager.createNativeQuery(nativeQuery, resultClass)` instead of `entityManager.createNativeQuery(nativeQuery)`). Without this option, we will return an object array. Only has an affect when using in conjunction with native query when consuming data. |
| `consumer.transacted` | `false` | **Camel 2.7.5/2.8.3/2.9: JPA consumer only:** Whether to run the consumer in transacted mode, by which all messages will either commit or rollback, when the entire batch has been processed. The default behavior (false) is to commit all the previously successfully processed messages, and only rollback the last failed message. |
| `consumer.lockModeType` | `WRITE` | **Camel 2.11.2/2.12:** To configure the lock mode on the consumer. The possible values is defined in the enum `javax.persistence.LockModeType`. The default value is changed to `PESSIMISTIC_WRITE` since **Camel 2.13**. |
| `consumer.SkipLockedEntity` | `false` | **Camel 2.13:** To configure whether to use NOWAIT on lock and silently skip the entity. |
| `usePersist` | `false` | **Camel 2.5: JPA producer only:** Indicates to use `entityManager.persist(entity)` instead of `entityManager.merge(entity)`. Note: `entityManager.persist(entity)` doesn't work for detached entities (where the EntityManager has to execute an UPDATE instead of an INSERT query)! |
| `joinTransaction` | `true` | **Camel 2.12.3:** camel-jpa will join transaction by default from Camel 2.12 onwards. You can use this option to turn this off, for example if you use LOCAL_RESOURCE and join transaction doesn't work with your JPA provider. This option can also be set globally on the `JpaComponent`, instead of having to set it on all endpoints. |
| usePassedInEntityManager | false | **Camel 2.12.4/2.13.1 JPA producer only:** If set to true, then Camel will use the EntityManager from the header<br><br>JpaConstants.ENTITYMANAGER instead of the configured entity manager on the component/endpoint. This allows end users to control which entity manager will be in use. |

| | | |
|---|---|---|
| sharedEntityManager | false | **Camel 2.16:** whether to use spring's SharedEntityManager for the consumer/producer. A good idea may be to set joinTransaction=false if this option is true, as sharing the entity manager and mixing transactions is not a good idea. |
| query | | To use a custom query. **Camel 2.19:** it can be used for producer as well. |
| namedQuery | | To use a named query. **Camel 2.19:** it can be used for producer as well. |
| nativeQuery | | To use a custom native query. You may want to use the option `resultClass` also when using native queries. **Camel 2.19:** it can be used for producer as well. |
| parameters | | This option is Registry based which requires the # notation. This key/value mapping is used for building the query parameters. It is expected to be of the generic type `java.util.Map<String, Object>` where the keys are the named parameters of a given JPA query and the values are their corresponding effective values you want to select for. **Camel 2.19:** it can be used for producer as well. When it's used for producer, Simple expression can be used as a parameter value. It allows you to retrieve parameter values from the message body header and etc. |
| resultClass | | Defines the type of the returned payload (we will call `entityManager.createNativeQuery(nativeQuery, resultClass)` instead of `entityManager.createNativeQuery(nativeQuery)`). Without this option, we will return an object array. Only has an affect when using in conjunction with native query. **Camel 2.19:** it can be used for producer as well. |
| useExecuteUpdate | | **Camel 2.19: JPA producer only:** To configure whether to use executeUpdate() when producer executes a query. When you use INSERT, UPDATE or DELETE statement as a named query, you need to specify this option to 'true'. |

## Message Headers

Camel adds the following message headers to the exchange:

confluenceTableSmall

| Header | Type | Description |
|---|---|---|
| `CamelJpaTemplate` | `JpaTemplate` | **Not supported anymore since Camel 2.12:** The `JpaTemplate` object that is used to access the entity bean. You need this object in some situations, for instance in a type converter or when you are doing some custom processing. See CAMEL-5932 for the reason why the support for this header has been dropped. |
| `CamelEntityManager` | `EntityManager` | **Camel 2.12: JPA consumer / Camel 2.12.2: JPA producer:** The JPA `EntityManager` object being used by `JpaConsumer` or `JpaProducer`. |

## Configuring EntityManagerFactory

Its strongly advised to configure the JPA component to use a specific `EntityManagerFactory` instance. If failed to do so each `JpaEndpoint` will auto create their own instance of `EntityManagerFactory` which most often is not what you want.

For example, you can instantiate a JPA component that references the `myEMFactory` entity manager factory, as follows:

xml<bean id="jpa" class="org.apache.camel.component.jpa.JpaComponent"> <property name="entityManagerFactory" ref="myEMFactory"/> </bean>

In **Camel 2.3** the `JpaComponent` will auto lookup the `EntityManagerFactory` from the Registry which means you do not need to configure this on the `JpaComponent` as shown above. You only need to do so if there is ambiguity, in which case Camel will log a WARN.

## Configuring TransactionManager

Since **Camel 2.3** the `JpaComponent` will auto lookup the `TransactionManager` from the Registry. If Camel won't find any `TransactionManager` instance registered, it will also look up for the `TransactionTemplate` and try to extract `TransactionManager` from it.

If none `TransactionTemplate` is available in the registry, `JpaEndpoint` will auto create their own instance of `TransactionManager` which most often is not what you want.

If more than single instance of the `TransactionManager` is found, Camel will log a WARN. In such cases you might want to instantiate and explicitly configure a JPA component that references the `myTransactionManager` transaction manager, as follows:

xml<bean id="jpa" class="org.apache.camel.component.jpa.JpaComponent"> <property name="entityManagerFactory" ref="myEMFactory"/> <property name="transactionManager" ref="myTransactionManager"/> </bean>

## Using a consumer with a named query

For consuming only selected entities, you can use the `consumer.namedQuery` URI query option. First, you have to define the named query in the JPA Entity class:

@Entity @NamedQuery(name = "step1", query = "select x from MultiSteps x where x.step = 1") public class MultiSteps { ... }

After that you can define a consumer uri like this one:

from("jpa://org.apache.camel.examples.MultiSteps?consumer.namedQuery=step1") .to("bean:myBusinessLogic");

## Using a consumer with a query

For consuming only selected entities, you can use the `consumer.query` URI query option. You only have to define the query option:

from("jpa://org.apache.camel.examples.MultiSteps?consumer.query=select o from org.apache.camel.examples.MultiSteps o where o.step = 1") .to("bean:myBusinessLogic");

## Using a consumer with a native query

For consuming only selected entities, you can use the `consumer.nativeQuery` URI query option. You only have to define the native query option:

from("jpa://org.apache.camel.examples.MultiSteps?consumer.nativeQuery=select * from MultiSteps where step = 1") .to("bean:myBusinessLogic");

If you use the native query option, you will receive an object array in the message body.

## Using a producer with a named query

For retrieving selected entities or execute bulk update/delete, you can use the `namedQuery` URI query option. First, you have to define the named query in the JPA Entity class:

@Entity @NamedQuery(name = "step1", query = "select x from MultiSteps x where x.step = 1") public class MultiSteps { ... }

After that you can define a producer uri like this one:

from("direct:namedQuery") .to("jpa://org.apache.camel.examples.MultiSteps?namedQuery=step1");

## Using a producer with a query

For retrieving selected entities or execute bulk update/delete, you can use the `query` URI query option. You only have to define the query option:

from("direct:query") .to("jpa://org.apache.camel.examples.MultiSteps?query=select o from org.apache.camel.examples.MultiSteps o where o.step = 1");

## Using a producer with a native query

For retrieving selected entities or execute bulk update/delete, you can use the `nativeQuery` URI query option. You only have to define the native query option:

from("direct:nativeQuery") .to("jpa://org.apache.camel.examples.MultiSteps?resultClass=org.apache.camel.examples.MultiSteps&nativeQuery=select * from MultiSteps where step = 1");

If you use the native query option without specifying *resultClass*, you will receive an object array in the message body.

## Example

See Tracer Example for an example using JPA to store traced messages into a database.

## Using the JPA based idempotent repository

In this section we will use the JPA based idempotent repository.

First we need to setup a `persistence-unit` in the persistence.xml file:{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-jpa/src/test/resources/META-INF/persistence.xml}Second we have to setup a `org.springframework.orm.jpa.JpaTemplate` which is used by the `org.apache.camel.processor.idempotent.jpa.JpaMessageIdRepository`:{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-jpa/src/test/resources/org/apache/camel/processor/jpa/spring.xml}Afterwards we can configure our `org.apache.camel.processor.idempotent.jpa.JpaMessageIdRepository`:{snippet:id=jpaStore|lang=xml|url=camel/trunk/components/camel-jpa/src/test/resources/org/apache/camel/processor/jpa/fileConsumerJpaIdempotentTest-config.xml}And finally we can create our JPA idempotent repository in the spring XML file as well:

xml<camelContext xmlns="http://camel.apache.org/schema/spring"> <route id="JpaMessageIdRepositoryTest"> <from uri="direct:start" /> <idempotentConsumer messageIdRepositoryRef="jpaStore"> <header>messageId</header> <to uri="mock:result" /> </idempotentConsumer> </route> < /camelContext> When running this Camel component tests inside your IDE
In case you run the tests of this component directly inside your IDE (and not necessarily through Maven itself) then you could spot exceptions like:

javaorg.springframework.transaction.CannotCreateTransactionException: Could not open JPA EntityManager for transaction; nested exception is <openjpa-2.2.1-r422266:1396819 nonfatal user error> org.apache.openjpa.persistence.ArgumentException: This configuration disallows runtime optimization, but the following listed types were not enhanced at build time or at class load time with a javaagent: "org.apache.camel.examples.SendEmail". at org.springframework.orm.jpa.JpaTransactionManager.doBegin(JpaTransactionManager.java:427) at org.springframework.transaction.

support.AbstractPlatformTransactionManager.getTransaction(AbstractPlatformTransactionManager.java:371) at org.springframework.transaction.support.TransactionTemplate.execute(TransactionTemplate.java:127) at org.apache.camel.processor.jpa.JpaRouteTest.cleanupRepository(JpaRouteTest.java:96) at org.apache.camel.processor.jpa.JpaRouteTest.createCamelContext(JpaRouteTest.java:67) at org.apache.camel.test.junit4.CamelTestSupport.doSetUp(CamelTestSupport.java:238) at org.apache.camel.test.junit4.CamelTestSupport.setUp(CamelTestSupport.java:208)

The problem here is that the source has been compiled/recompiled through your IDE and not through Maven itself which would enhance the byte-code at build time. To overcome this you would need to enable dynamic byte-code enhancement of OpenJPA. As an example assuming the current OpenJPA version being used in Camel itself is 2.2.1, then as running the tests inside your favorite IDE you would need to pass the following argument to the JVM:

-javaagent:<path_to_your_local_m2_cache>/org/apache/openjpa/openjpa/2.2.1/openjpa-2.2.1.jar

Then it will all become green again 😅

Endpoint See Also

- Tracer Example